Island Hopping: Efficient Mobility-Assisted Forwarding in Partitioned Networks

Natasa Sarafijanovic-Djukic, Michał Piórkowski, and Matthias Grossglauser
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

***Abstract*— Mobile wireless ad hoc and sensor networks can be permanently partitioned in many interesting scenarios. This implies that instantaneous end-to-end routes do not exist. Nevertheless, when nodes are mobile, it is possible to forward messages to their destinations through mobility.**

We observe that in many practical settings, spatial node distributions are very heterogeneous and possess concentration points of high node density. The locations of these concentration points and the flow of nodes between them tend to be stable over time. This motivates a novel mobility model, where nodes move randomly between stable islands of connectivity, where they are likely to encounter other nodes, while connectivity is very limited outside these islands.

Our goal is to exploit such a stable topology of concentration points by developing algorithms that allow nodes to collaborate to discover this topology and to use it for efficient mobility forwarding. We achieve this without any external signals to nodes, such as geographic positions or fixed beacons; instead, we rely only on the evolution of the set of neighbors of each node.

We propose an algorithm for this *collaborative graph discovery* problem and show that the inferred topology can greatly improve the efficiency of mobility forwarding. Using both synthetic and data-driven mobility models we show through simulations that our approach achieves end-to-end delays comparable to those of epidemic approaches, while requiring a significantly lower transmission overhead.

Keywords: delay-tolerant networks; partitioned networks; mobility; routing.

I. Introduction

In many applications of wireless ad hoc and sensor networks, the network is frequently or permanently partitioned, i.e., end-to-end routes between some pairs of nodes do not exist. Such scenarios include large-scale emergency and military deployments without fallback infrastructure, environmental monitoring [1], transportation networks [2], self-organized "pocket-switched" networks [3], etc. These networks may be partitioned because of subcritical node density, channel fluctuations (shadowing and fading) and node mobility.

Although *instantaneous* end-to-end routes do not always exist, it is often the case that a message can nevertheless be delivered over time, where the message has to be temporarily buffered at intermediate nodes to wait for the availability of the next link towards the destination. The problem of finding a route over time amounts to finding a sequence of mobile nodes that physically carry the message towards the destination. This approach has been referred to as *mobility-assisted forwarding*, or also as *store-carry-forward* [4], [5]. Finding such routes through space and time is obviously a complex problem in general, and depends heavily on the joint statistics of link availability [6].

In this paper, we are interested in the case where network partitions arise because the distribution of nodes in space is heterogeneous. Specifically, we assume that the network possesses concentration points (CPs), i.e., regions where the node density is much higher than on average, and where nodes have therefore a much better chance of being connected to other nodes than on average. We believe that many real networks possess such concentration points, even though most network models assume homogeneous node distributions for convenience and tractability [7], [8].

Furthermore, we argue that the CPs, and the average flows of nodes between CPs, typically remain stable over relatively long time-scales. This is because they depend on features of the natural or constructed environment, which change over time-scales much longer than the delivery time of messages.

Our goal is to develop efficient schemes for mobility-assisted forwarding that take explicit advantage of the presence of stable concentration points. To achieve this goal, we make three distinct, but strongly related contributions: (i) we introduce a mobility model that explicitly embodies CPs, and justify it through an analysis of a large mobility trace; (ii) we describe the Island Hopping (IH) algorithm that forwards messages through mobility; and (iii) we describe how a collection of mobile nodes can infer the CP topology without any explicit signals from the environment, such as GPS coordinates or beaconing signals. Finally, we summarize these three contributions.

A. Mobility Model with Stable Concentration Points (CPs)

Our first contribution is a new mobility model that embodies the presence of stable CPs. This model is pessimistic, in the sense that we assume that nodes are only able to communicate with other nodes at the same CP; outside these islands of connectivity, they are not able to communicate. We therefore view the network as a graph $G(V, E)$, where the vertex set V represents the CPs, and the edge set E represents flows of mobile nodes between the CPs. Two nodes can communicate with each other only if they are at the same vertex.

Note that the only assumption we make about the set of nodes within a same CP is that they form a connected subgraph, i.e., that each node can reach each other node. If all the nodes are within radio range of each other, then this is straightforward; if not, then a message sent between two nodes at a same CP may have to traverse multiple intermediate hops. Our routing and graph discovery algorithms rely on simple broadcast primitives within a CP. They can be implemented through physical-layer broadcast, or through flooding algorithms.

Results from Large Mobility Trace. We provide some evidence from a large-scale dataset to justify the validity of our mobility model. We analyze a trace of the movements of $n \approx 800$ taxis over a three-month period in the city of Warsaw, Poland. We find that in some areas of the city the expected number of cars within radio range of each other is much higher than elsewhere. Furthermore, we find that these

areas are stable over time, in the sense that an area that sees a high density of cars on a particular day is likely to see a high density on another day as well. This justifies our assumption of a stable topology of CPs in one realistic scenario.

B. Mobility-Assisted Forwarding in the CP Model

Our second contribution is a novel mobility-assisted forwarding algorithm called Island Hopping (IH). This algorithm explicitly exploits knowledge of the CP graph. It forwards a message through a sequence of CPs to its destination. Each CP represents an opportunity to pass the message to other nodes.

The key question is how to pass a message from one CP to the next CP through nodes whose future movements are random and unpredictable. If the future movements of nodes were known, we could pass the message to a single node that would move in the right direction, i.e., to a CP closer to the destination in $G(V, E)$. However, given that future movements are unpredictable, the IH algorithm makes a small number of copies of a message at each CP, in the hope that at least one copy will move to the intended next CP and the other other copies will be discarded. The process repeats at the next CP, until the message reaches its destination. The key challenges are to (i) not lose the message completely, and (ii) to avoid that an unnecessarily large number of copies are generated.

C. Collaborative Graph Discovery (COGRAD)

Our third contribution is a distributed algorithm that allows the nodes to collaboratively discover the CP graph, *in the absence of any signal from the environment*, such as GPS coordinates or fixed beacons. This is important, because it would be unrealistic to assume that the graph of CPs and the flows of mobile nodes between CPs is known a-priori. Instead, we assume that the only information that nodes have available is the set of other nodes that they can reach (either directly or over multiple hops), which can be discovered in a straightforward manner (hello messages, flooding, etc.)

In a nutshell, the COGRAD algorithm achieves this in two phases: vertex labeling and edge discovery.

Vertex Labeling. The goal of this phase is to generate a label, i.e., a unique identifier, for each vertex of V, that will remain stable over time, even though nodes move in and out of each vertex. Suppose that at a given time the nodes currently located at the same CP agree on a label for this vertex. Now another node i arrives at this vertex. Node i has not received any explicit clue from the environment that it has moved, and the other nodes have not received a clue that they have not moved. However, node i's set of neighbors has changed rather markedly, whereas the other nodes' neighbor set has only seen the addition of i. These nodes can therefore decide jointly that it is likely that node i has moved, and the other nodes have not; node i therefore accepts the label of this vertex.

This process associates labels with vertices. The labels remain stable even though the set of nodes at a vertex changes all the time. Although errors can occur in this process (e.g., if a vertex becomes completely empty for a while), this does not affect the performance of the routing algorithm if this occurs rarely.

Edge discovery. Once we have associated a label with each island, a node discovers the edges of the CP graph as it moves from island to island. We also let nodes exchange edges they have discovered through a gossip protocol. This ensures that each node learns the entire graph, even though it may only visit part of the graph. Also, this ensures that outdated information (labels that have become invalid after errors) is flushed out.

To summarize, we argue that stable concentration points are prevalent, and that they can be exploited for efficient mobility-assisted forwarding. We present evidence to this effect, and a mobility model embodying CPs, in Section III. In Section IV, we then describe the routing algorithm for mobility-assisted forwarding in the presence of stable CPs. For simplicity of presentation, we describe this algorithm assuming that the CP graph, and node positions on the graph, are known. In Section V, we describe *collaborative graph discovery (COGRAD)*, a distributed algorithm that infers the CP graph from each node's dynamic neighborhood set. This allows the routing algorithm to operate without any explicit clue to nodes about their location and movement. In Section VI, we show extensive simulation results on synthetic graphs, as well as on the graph derived from our traffic dataset described earlier. We show that our IH algorithm in conjunction with COGRAD results in a scheme that achieves delay of the order of much more aggressive flooding-based schemes, while requiring a much smaller number of copies of each message. We conclude the paper in Section VII.

II. Related Work

Routing in partitioned networks has been investigated in two scenarios: 1) when the dynamics of connectivity between nodes is known in advance ([6]), and 2) when it is unpredictable ([5], [9], [10], [11], [12], [13]).

The latter can be further classified as controlled mobility and random mobility. Under controlled mobility [5], node trajectories can be controlled and adapted.

Forwarding algorithms for random mobility are usually based on some form of flooding. For example, in epidemic routing (ER) [9], when two nodes meet they exchange all messages that only one of them has a copy of. In this way, a message is essentially flooded to all nodes, which ensures that it will reach the destination. This flooding can be constrained by taking into account mobility history of nodes, e.g., patterns of encounters of nodes, but also other parameters, such as the energy left at the node.

PROPHET [10] is based on ER, where message exchanges between nodes take into account a probability that a node will meet the destination of a message in the future. This requires the exchange of vectors of estimated probabilities. In DTC [11] a node carrying a message checks periodically whether to transfer the message. The message is transferred if in the set of nodes (possible via multihop) connected to this node there is a node "closer" to the destination. "Closeness" depends on mobility history, system parameters, and future mobility (if available). A mathematical framework for calculating this "closeness" based on utility functions is given in [12]. In MDDV [13], forwarding decisions in a vehicular network are made based on knowledge of the road map, where nodes are equipped with GPS receivers.

In the Spray and Wait algorithm by Spyropoulos et al. [14], the number of transmissions of a message is constrained by letting only a fixed number L of copies of a message to be "sprayed" into the network. Then these copies "wait" until they meet the destination. The authors show that the parameter L controls the tradeoff between end-to-end delay and overhead.

Our approach considers random unpredictable mobility and distinguishes itself in that we explicitly *infer and use spatial information* of node mobility to limit flooding.

Most mobility models give rise to homogeneous node distributions over a two-dimensional area [7]. These models lack an important feature of realistic mobility patterns, the fact that nodes often cluster around preferred areas [15]. Hsu et al. [16] propose a realistic Weighted Way Point (WWP) mobility model that incorporates the fact that destinations are not uniformly selected from the simulation area. The authors designed the WWP model as a time-varying Markov model, fitted to data from a survey on the USC campus. Tang and Baker in [17] analyze mobile traces with 24773 radios in a metropolitan-area network. They observe a significant clustering of radios in some areas, e.g., a financial district; however, they do not attempt to model user mobility.

III. A Mobility Model for Heterogeneous Partitioned Networks

In this section, we introduce and motivate a new mobility model for partitioned networks. In particular, we focus on one feature that appears to be quite ubiquitous in real mobility processes: concentration points (CPs), i.e., regions where mobile nodes have a much higher chance of encountering other nodes than elsewhere. Examples of CPs include:

- People in urban environments: workplace, restaurants, public transportation (train stations, airports), movie theaters, etc.
- Wildlife monitoring: watering holes, clearings, oases, etc.
- Office buildings: cafeterias, conference rooms, water coolers, hallways, etc.
- Road traffic: intersections, parking lots, gas stations, traffic lights, etc.
- Military: bases, camps, forts, ports, etc.

In the next subsection we use a large data set of GPS coordinates of a taxi fleet collected during a three-month period to verify the presence and the stability of CPs.

A. Stable Concentration Points in a Mobility Trace

Here we analyze the mobility traces of taxi cabs from the MPT Radio Taxi company from Warsaw, Poland.[1] This data set contains GPS coordinates of 825 taxis collected over 92 days in an area of 60 x 48 [km]. Each taxi sends a *location update* (timestamp, identifier, GPS coordinates) to a central server. Updates are not periodic but irregular - they can be as frequent as a few per hour but also as a few per day.

We want to check whether this data set confirms the existence of stable CPs. We divide the whole area into a grid of cells of equal size. For each day d and each cell (x, y) we find the *normalized taxi population* - $f(x, y; d)$, interpreted as the empirical probability that a random update falls into the cell (x, y) on day d (cf. Figure 1). We look at two different sizes of a cell cell is possible using the 802.11-like wireless device. within which a direct radio communication is possible. Based on statistical analysis of the data set we conclude that:

1) Spatial distribution is heavy tailed: Figure 2 shows the empirical complementary cumulative distribution function (CCDF) of $f(x, y; d)$ for the entire data set. This distribution has a heavy tail, which implies that some cells have population density much above the average. We show later in this paper (Section VI) that our new routing scheme requires that the average number of nodes at a CP is at least 15 or so. The heavy-tailed distribution in Figure 2 also implies that the number of CPs satisfying this requirement will vary relatively slowly with the total number n of nodes. For example, if we scale up our data set, assuming a total of 3 million different vehicles (private cars, public transportation, company vehicles, trucks, etc.) in this city, we estimate that there are about 89 different cells of size 480x480 meters for which the expected number of vehicles per day is larger than 15.

[1] http://www.taximpt.com.pl

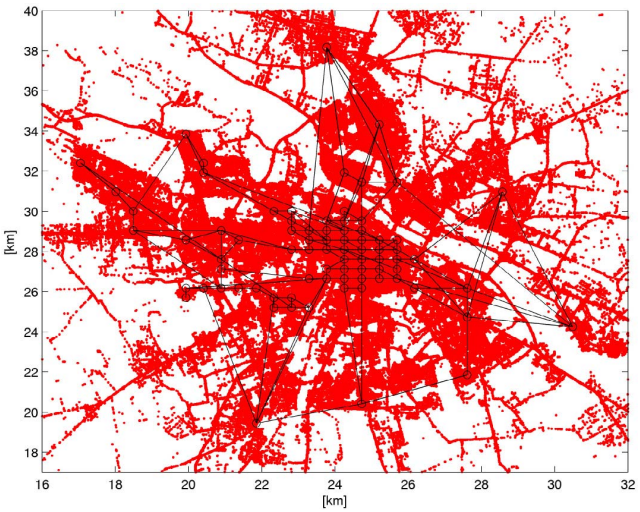


Fig. 1. The red dots represent superimposed location updates of 825 taxis over 92 days taken from the data set. The black circles represent 79 CPs and black lines show taxi flows between these CPs. Both CPs and flows are extracted from the data set.

2) Spatial distribution is stable over time: We expect that the high population cells remain stable over time. Figure 2 inset shows also a scatter plots of $f(x, y; d)$ for one randomly chosen pair of days (d_1, d_2). We observe significant clustering along the diagonal, which means that the spatial distribution on different days tends to be strongly correlated. Furthermore, we observe that the more densely populated cells (upper-right quadrant) tend to be particularly close to the diagonal, which is a good visual confirmation of our hypothesis. In addition to this visual test, we also apply the Kruskal-Wallis statistical test [18] to verify whether the normalized taxi populations for each cell on different days come from the same distribution[2]. This is a non-parametric test that makes no assumptions about the distribution of the data. We find that the p-value of the test is 0.04. For the size of test $\alpha = 0.01$ this results in a high confidence in our hypothesis.

B. CP Graph

Given the above observations, we now define a mobility model that embodies CPs. The network topology is given by a directed connected graph $G(V, E)$ whose vertex set V represents the CPs, and whose edges describe the possible movements of nodes between CPs. There are n nodes that move on this graph. At every time t, every node i is either located at one CP, or is en-route between two CPs u and v. We denote the current position of node i by $X_i(t)$. We assume that time is continuous. We call $B_i(t)$ the set of neighbors of node i at time t (including itself), i.e., the set of nodes located at the same CP as i. If a node is en-route between two CPs then $B_i(t) = \{i\}$.

[2] We exclude from the data set all Fridays and Sundays because we observe that these days differ much from the other days. This is probably due to people moving out from the city for the weekends.

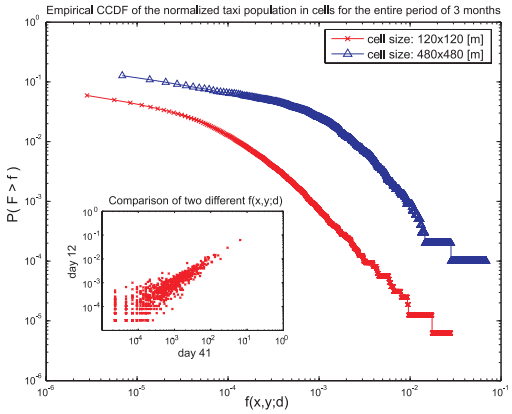


Fig. 2. Empirical CCDF of $f(x, y; d)$ for the entire period for two levels of discretization. Inset shows the scatter plot of $f(x, y; d)$ on two random days - each point on the plot corresponds to a density in a cell (x, y) at days d_{12} and d_{41}.

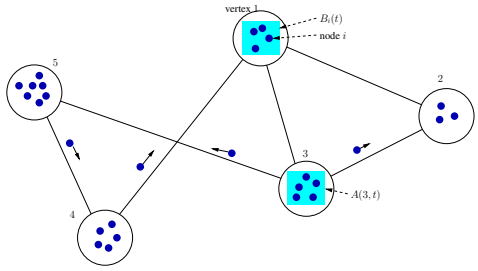


Fig. 3. Nodes move on a graph $G(V, E)$, which describes the network topology in terms of its CPs and the ways nodes can move between them.

We assume that nodes located at the same CP can communicate with each other (either directly or through multi-hop), whereas nodes at different CPs cannot. We assume that the only observation available to a node about its environment is the set $B_i(t)$, which the node can obtain by communicating with other nodes.

C. Inferring the CP Graph from Mobility Trace

In this section we describe our heuristic to extract a CP graph from the mobility trace.

1) Concentration Points: In order to find CPs we first define a *cluster* at day d as a cell for which $f(x, y; d) > 5\%$. The reason we choose 5% as the threshold is to ensure that in every CP there are at least 15 vehicles (see III-A.1). Using such a small threshold allows us to identify clusters that would be more visible for regular and frequent location updates. We identify 174 clusters. Here we define a CP as a cluster that is present for more than 20 days (see III-A.2). In result we find 79 CPs within the whole city (cf. Figure 1).

2) Stable Flows: Inferring stable flows between CPs from such data set is not trivial because of very irregular and infrequent location updates. Suppose a taxi goes from u to v only through w and it sends only *two* updates: at time t_1 from u and at time t_2 from v. Thus by looking only at the travel time $\tau(u, v) = t_2 - t_1$ we could conclude that there is a direct edge from u to v, which is not true. This would lead to a CP graph that does not reflect the topology of the city. In this section we propose a heuristic that solves this problem. First, we enumerate the entire data set looking for the *minimum travel time* for each pair of CPs: $\tau_{min}(u, v)$. If $\tau_{min}(u, v) > 0$ then the flow between u and v exists, i.e., $e = (u, v) \in E$. In result we get a CP graph that is close to fully connected. In order to prune the unrealistic edges we use the following heuristic: if $\exists p(u, v) = (u, w_1, w_2, ..., v)$ s.t. $\tau_{min}(u, v) > \sum_{e \in p(u,v)} \tau_{min}(e)$ then $(u, v) \notin E$, where $p(u, v)$ is the path between u and v. Instead of almost fully connected graph, we get a graph for which the average vertex degree is 3.2.

The inferred CP graph is shown on Figure 1. The resulting topology resembles a spider net, which is consistent with the topology of the city of Warsaw, where most of the important institutions and centers of activity are located downtown. Moreover, knowing the real distances between all CPs and the $\tau_{min}(e)$, we find that the median value of the maximum speeds found over the edges of the CP graph is 58.1 km/h, which is a reasonable value.

IV. Island Hopping (IH) Algorithm

Island Hopping (IH) is a mobility-assisted forwarding algorithm in which a node makes forwarding decisions, i.e., when to pass a copy of a message to other nodes and when to discard it, by using the knowledge of:

1) the CP graph, and its own position in that graph, and
2) the destination's position in the CP graph.

The design goals are to minimize the number of copies made of a message, to minimize the end-to-end delay, and to maximize the delivery rate.

In this section we describe our IH scheme under the assumption that a node has knowledge of the CP graph G and its own position in G at all times. Inferring this knowledge in scenarios where nodes have external signals from the environment (such as GPS coordinates or signals from fixed beacons) is easier than in the case where no such external information exists. In Section V, we show how nodes can infer this knowledge without such external signals.

A. Message Progression Towards a Fixed Destination

In this subsection, we show the main ideas of IH under the further simplifying assumption that nodes know the position in G of a message's destination. In the next subsection, we then show how nodes can locate the destination.

Our IH scheme uses the following three ideas, which we illustrate in Figure 4.

1) Routing a Message through a Sequence of CPs: Assume that a node i, currently located at vertex $u \in V$, has a message m with destination node D located at vertex $w \in V$. The key is for node i to decide which vertex v should be the next hop in V for message m in order to make progress towards the destination. This desired next hop v is stored in the message in the field $m.next\_hop$. We choose this next hop v as a neighboring vertex on the shortest path between vertices u and w in the CP graph.

The next move of node i is in general not yet known. If node i happens to move to the desired next hop v, then node i keeps m, and generates new copies in other nodes. If node i moves to another vertex $v' \neq v$, node i discards m.

In Figure 4(a), node S at vertex 1 originates a message m to node D at vertex 4. Node S makes several copies of m with $m.next\_hop := 2$. Figure 4(b) shows what happens when these copies move to neighbouring vertices. The node with the copy that moves to vertex 3 discards m because $m.next\_hop \neq 3$, whereas the node with the copy that moves to vertex 2 makes new copies of m with $m.next\_hop := 4$. This process continues until the message reaches node D at vertex w.

2) At Least One Copy Moves to the Next-hop CP: If none of the copies of message m move to $m.next\_hop$, then all these copies of m will be eventually discarded, and m will be lost. To boost the probability that at least one copy of m progresses towards the next-hop vertex, we introduce a "one-hop" acknowledgement (ACK) scheme. The goal of this scheme is to piggyback one-hop delivery information about message m through nodes moving in the reverse direction, and to generate additional copies if needed.

Assume that there exist copies of m at vertex u with $m.next\_hop = v$. Nodes at vertex u should be informed when a copy of m has reached v. When a node with m arrives at v, it broadcasts this fact to all nodes at v. If one of these nodes then moves to u, it broadcasts an ACK for m. All nodes at u can then discard m. But if a node at u holding m has not received an ACK by the time where only a small number c_1 of copies of m are left, it generates additional copies of m. This process repeats for at most c_2 times.

In Figure 4(b), a node with m moves from vertex 1 to vertex 2, where it generates new copies, and broadcasts to all nodes the identity of m. In Figure 4(d), one of these nodes arrives at vertex 1 and broadcasts an ACK for m. Then all copies of m at vertex 2 are discarded.

3) Only One Copy Survives to the Next-hop CP: If more than one copy of m with $m.next\_hop = v$ moves into vertex v, then new copies of m can be generated at v several times. This could lead to an exponential increase of the number of copies. We include a mechanism to suppress additional rounds of copying. If node i moves to v, it makes new copies only if none of the nodes currently at v have seen an earlier copy of m arrive at v.

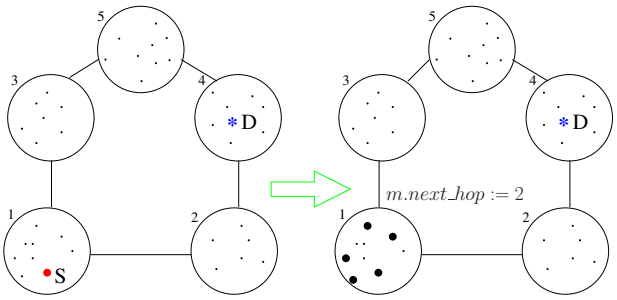
Figure 4(c) shows what happens when a second copy of m arrives at vertex 2 from vertex 1. Even if $m.next\_hop = 2$, the copy is discarded, because m has already been at vertex 2.

B. Dynamically Locating Destination through Last Encounter Routing

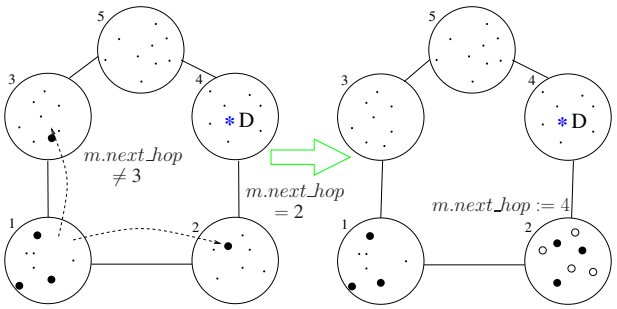
So far, we have assumed that the location of the destination is fixed and known to the message, which is unrealistic. To discover the location of the destination of a message, we cannot resort to the classical methods such as flooding, because the network is partitioned.

To solve this problem, we borrow an approach from [19] called Last Encounter Routing (LER), where a node maintains a Last Encounter Table (LET), with an entry for every other node. An entry consists of the time and location of its last encounter with the node. In [19], the location of the node is its geographic location. We adapt this to our setting, where the location of last encounter is a vertex: each node remembers for each other node the last time they were located at the same vertex, and therefore connected.

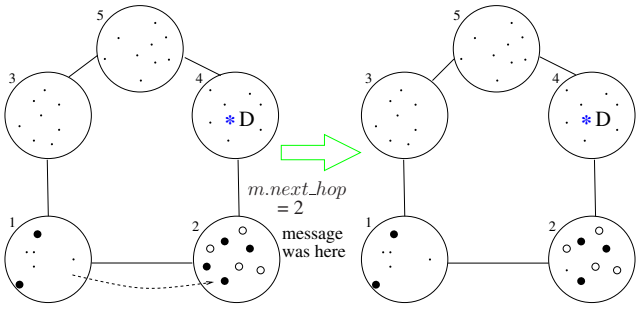
The LETs are used by a message to continually obtain more recent information about the location of the destination, as follows. Assume again that a node i at a vertex u has a message m destined for a node D. As we saw in Section IV-A, node i needs to determine the next-hop vertex for m. Before doing so, node i searches all nodes at u for the most recent LET entry for node D. This location is then used as an estimate of the position of node D to determine the next-hop vertex. The message remembers this estimate in a field $m.le$. As the message gets closer, it tends to find more recent information, "zeroing in" on the destination.



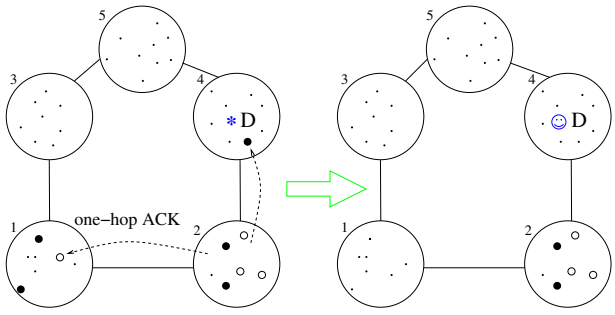
(a) Node S originates a message m to node D.



(b) Node with a copy of m moves to another vertex.



(c) Node with a copy of m moves to a vertex where m has already been observed.



(d) One-hop acknowledgment.

Fig. 4. Island hopping - example.
Small dots - nodes without a copy of m; large dots - nodes with a copy of m; empty dots at a vertex - nodes without a copy of m, but that know that m was at this vertex.

V. Collaborative Graph Discovery (COGRAD)

We now specify how *collaborative graph discovery (COGRAD)* infers the CP graph from changes in neighborhood sets, without any other signal from the environment. Note that a single node would obviously be unable to find out anything about the network topology; a COGRAD protocol is necessarily collaborative. More formally, the goal of COGRAD is:

- for every node to learn the CP graph $G(V, E)$;
- for every node i to know its current position $X_i(t)$ at all times t.

As we mentioned in Section I, COGRAD achieves this by having the nodes running two algorithms: vertex labeling and edge discovery. The vertex labeling algorithm decides for each node if this node is currently on an edge (i.e., in transit between two vertices) or at a vertex (CP); in the latter case, it also identifies the CP through a label. The node then uses the output of this algorithm as input into the edge discovery algorithm. The edge discovery algorithm estimates the edge set E of the CP graph. We next describe these two algorithms.

A. Vertex Labeling

We have assumed that if node i is at an edge, then its neighborhood set is $B_i(t) = \{i\}$; at a vertex, the neighborhood set includes the other nodes present at the same vertex. Using this assumption, a node i can decide whether it is at an edge or at a vertex: if $|B_i(t)| = 1$, it concludes that it is at an edge, otherwise it concludes that it is at a vertex[3].

If a node is at a vertex, it decides on a label for this vertex. Let us now see how nodes label vertices. At startup, nodes that are at a vertex agree on a label for this vertex, e.g., by taking a random number from a large alphabet, or by taking the minimum node identifier of nodes at the vertex. This ensures an initial set of unique vertex labels.

As a node moves from a vertex u through an edge to another vertex v, it should accept the label of the new vertex v as its new location. Assume that node i arrives at v at time t. Let us denote with t^- the time immediately before t. If nodes at v at time t^- have already decided on a label, then node i simply accepts this label (cf. Figure 5(a)). If not, then node i randomly generates a new label (cf. Figure 5(b)).

Algorithm 1 formally describes the vertex labeling scheme described above. A node runs the algorithm every time its neighborhood set changes. With $Y_i(t)$ we denote node i's position in the CP graph obtained by the algorithm at time t, where $Y_i(t) = edge$ if node i thinks it is at an edge, and $Y_i(t) = vertex\ y$ if node i thinks it is at vertex with label y. With random we denote a large random integer.

Algorithm 1: Labeling - event: $B_i(t^-) \neq B_i(t)$, $t > 0$

```
If |B_i(t)| = 1 then /* node i alone */
    Y_i(t) = edge
Else /* node i not alone */
    If |B_i(t^-)| = 1 then /* node i alone before */
        If |B_i(t)| > 2 then /* at least 2 other nodes at this CP /*
            Y_i(t) = Y_j(t^-), for some j ∈ B_i(t) /* accept new label */
        Else /* only 1 other node at this CP */
            Y_i(t) = vertex random /* relabel this CP */
        End if
    Elseif /* node i not alone before */
        Y_i(t) = Y_i(t^-) /* keep current label */
    End if
End if
```

[3]Note that the assumption that a node is alone if it is at an edge can be relaxed in a realistic setting by adding a threshold for the size of the set of neighbors when a CP becomes a vertex in the CP graph.

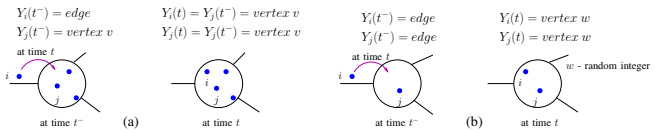


Fig. 5. Vertex labeling when a node arrives at a vertex. With $Y_i(t)$ we denote node i's position in the CP graph obtained by the vertex labeling algorithm at time t.

Relabeling Errors. Once a vertex is labeled, nodes at this vertex keep the same label, until there is only one node left at the vertex. Then this remaining node falsely decides that it is at an edge. The next time two nodes are present at this vertex, the vertex is relabeled. Therefore, a vertex changes its label every time the number of nodes drops below two. If this occurs rarely, then the vertex labeling algorithm manages to maintain stable vertex labels over relatively long periods of time.

B. Edge Discovery

As a node moves on the graph, it observes the label of every vertex it visits, as described in the previous section. If a node moves directly from vertex u with label y_u to vertex v with label y_v, then this indicates the existence of a labeled edge (y_u, y_v), and we say that the node *directly observes* this edge.

To discover the edge set E, it would be possible for each node to rely only on its own observations of the edges it traverses. However, this approach has the following drawbacks. First, if node mobility is such that a node does not visit the entire graph, then this node will never discover some parts of the graph, which can result in poor forwarding decisions. Second, even if a node moves over the entire graph, the discovery process would be rather slow, and the transient time (until every node knows most of the CP graph) would be excessively long.

We therefore would like to accelerate the dissemination of edge information to allow every node to learn the entire graph. One approach is for nodes to exchange labeled edges through a gossip protocol. This allows nodes to learn the entire CP graph more quickly.

Note however that a node's view of the CP graph may change over time because of relabeling errors. If the label of a vertex u changes from y_u to y'_u, then all edges with label y_u become obsolete. We use an aging mechanism to eliminate such obsolete edges.

More precisely, in our gossiping scheme, node i's view of the CP graph, G_i, is represented by the set of pairs (e, t_{obs}), where t_{obs} is the time when edge e was directly observed. Node i has edge e in G_i either if it has directly observed e, or if it has received e through gossiping from other nodes.

Every node, upon arrival at a vertex, gossips a constant number (n_e) of randomly chosen entries (e, t_{obs}) from its view of the graph G_i to all nodes at this vertex.

Therefore, node i updates G_i either when it directly observes an edge, or when it receives a gossip message from another node. When node i directly observes edge e at time t, it adds (e, t) to G_i and deletes the old entry for e from G_i if it exists. When node i receives (e, t_{obs}) through gossiping, it does the following. If it does not have an entry for e in G_i

yet, then it adds (e, t_{obs}) to G_i. If it already has an entry for e, (e, t^i_{obs}), and if $t_{obs} > t^i_{obs}$, then it replaces (e, t^i_{obs}) in G_i with (e, t_{obs}).

In addition to the process of learning edges, a node removes an edge from the graph if the edge grows too old. More precisely, a node removes entry (e, t_{obs}) from its graph at time $t_{obs} + T_{age}$, where T_{age} is a fixed constant for all nodes.

C. IH and COGRAD

In the previous section, we had described the IH algorithm as operating on top of a topological oracle that reveals the CP graph G to every node. We now describe how IH operates on top of COGRAD, where the CP graph G, and each node's position on the graph, are obtained through COGRAD, and therefore subject to some errors, as described.

As we saw in Section IV, a node i located at vertex u chooses the next hop for a message m with the destination at w as the next vertex on the shortest path $u \rightarrow w$ in its view of the CP graph. We have not specified which next-hop is chosen if several shortest paths, and thus several possible next-hops, exist. In this case, we use the age of edges obtained by COGRAD in the edge discovery part to choose between these possible next-hops. We choose the next-hop with the smallest age of its incoming edge from u, i.e., the vertex v whose entry $((u, v), t)$ is most recent.

We discuss two other error conditions that can arise in IH due to errors in the underlying graph discovery algorithm. The first situation arises when node i makes the next-hop decision for a message, but cannot find a shortest path to w. This can happen when in i's view of the graph G_i, (i) w is not known, or (ii) a path $u \rightarrow w$ does not exist in G_i, possibly because u has no outgoing edges in G_i. The second situation arises when the destination location (obtained by the last encounter tables as described in Section IV) is the current node's location u, but the destination is elsewhere.

We resolve these situations as follows. In the case where a node is not aware of any outgoing edges from the current vertex u, then this node simply keeps the message and waits until it moves to another vertex v, where it then tries again to find a path. Otherwise, the node chooses as next hop the vertex v for which it has the most recent entry $((u, v), t)$ in its edge cache.

VI. Performance Evaluation

In this section, we evaluate the performance of the IH algorithm, both in combination with an oracle that reveals the true vertex or edge identity to each node, and in combination with the COGRAD algorithm that discovers the CP graph and node locations from each node's neighborhood set $B_i(t)$, as described in Section V. For this purpose, we developed a custom simulator implementing the CP graph-based mobility model, and the IH and COGRAD algorithms.

We compare our algorithm with ER [9] and PROPHET [10]. We also compare the delay of our algorithm with the scheme where a source transmits a message only to the destination (no routing algorithm is used). We denote this scheme as "no R".

A. Simulation Set-up

We present simulation results for both synthetic topologies and for the city traffic topology we have inferred in Section III. The synthetic topology we use is the 2-dimensional $k \times k$ grid, where we vary k from 3 to 10, i.e., $|V| = 9, \ldots, 100$.

For both the synthetic and the city topology we assume that we have $n = c|V|$ nodes performing independent random walks on the graph, with $c > 0$ a parameter controlling the mean number of nodes per vertex. Note that the random walk is the most challenging mobility process for our purposes, for the following reason. When a node performs a random walk, its future movements are independent of the entire past. In other words, even if a node accumulates statistics about its past movements, this will not help predict the future. As such, all the nodes located at a vertex v at a given time t are statistically equivalent, and no information about the past (e.g., keeping track of when a node has last seen the destination, as in PROPHET) can be used to predict where a node will go in the future.

The location of a node is either a vertex $v \in V$ (which implies that the node can communicate with all other nodes currently located at the same vertex v), or on an edge $e = (u, v) \in E$ (which implies that the node is en route from island u to island v, and is not able to communicate with any other node).

Each node spends an exponentially distributed time with mean T_V at a vertex, and an exponentially distributed time T_E at an edge, where we set $T_V = 10T_E$. All the delay results we report, as well as all time scales, are normalized by setting $T_V + T_E = 1$, i.e., we normalize to unity the average speed at which a node advances from vertex to vertex.

So far, the only assumption we make about the set of nodes within a same CP is that each node can reach other nodes (either because they are in direct radio range of each other, or because they can form a connected ad hoc network). However, for the purpose of simulations we do assume for simplicity that nodes within a CP are directly connected. This assumption is favorable for both our algorithms and for the epidemic-based algorithms ([9],[10]) we compare with, because it makes it possible to transmit a message to all nodes at a CP in a single broadcast. If nodes at a CP were not directly connected, then the broadcast function would have to be replaced by a flooding primitive.

Other fixed parameters in our simulations are: $c_1 = 3$ and $c_2 = 2$ in IH ($c1$ and $c2$ are defined in Section IV), and $maximum\ hop - count = 1000$ in ER and PROPHET.

Each simulation we report is preceded by a warm-up phase, which is needed to populate the last encounter tables (LETs). The warm-up phase terminates if 80% of the node pairs have encountered each other at least once; note that this is conservative in that the LE tables are asymptotically fully populated.

We use the following metrics:

- delivery rate - the number of messages delivered to destinations divided by the total number of messages sent by sources
- delay - the normalized delay for the delivered messages
- number of transmissions per message - how many times a message is transmitted until there are no more copies of this message in the network.

We find these metrics by averaging over a number of randomly chosen source-destination pairs, where for each pair a source sends a single message to its destination. In simulation results we show the mean values of these metrics with 95% confidence intervals.

Note that the number of transmissions per message includes only the transmissions of actual messages, not the control

messages generated by IH and COGRAD. We justify this as follows. First, in the IH algorithm, control messages result only when a traffic message is to be transmitted. Therefore, the IH control overhead should be considered relative to the overhead to transmit messages. Usually, data messages tend to be orders of magnitude larger than control messages. This is in compliance with the proposed architecture for delay tolerant networks in [4]. Therefore, the IH overhead per data message can be neglected. Second, in the COGRAD vertex labeling algorithm, control overhead accrues when a node discovers its neighborhood set $B_i(t)$, and this type of control overhead is also present in ER and PROPHET. Third, in the COGRAD edge discovery algorithm, control overhead accrues when a node arrives at a vertex and broadcasts a small number of edges. In comparison, ER and PROPHET exchange a summary vector and predictability vectors whenever a node meets a new node, which in our setting means the broadcast of these vectors per each node's arrival at a vertex. Therefore, the control overhead in our scheme is comparable to that in ER or PROPHET, and small compared to data messages except when the network is very lightly loaded. Hence, we consider only traffic messages in our simulation results and we leave a detailed analysis of the control overhead for future work.

B. Simulation Results

First, we show in Figures 6 and 7 how these metrics depend on the two parameters of the COGRAD algorithm (n_e and T_{age}, c.f. Section V). Recall that the parameter n_e represents the number of edges that each node gossips upon its arrival at a vertex, and the parameter T_{age} is the age threshold fixed for all nodes upon which an edge is removed from the CP graph.

We see that the results are not very sensitive to T_{age}, i.e., that in the broad range of T_{age} from $50(T_V+T_E)$ to $100(T_V+T_E)$ the results vary little. We also see that the good results are achieved with a relatively small number of gossiped edges ($n_e = 4$), especially in the case of the grid topology. In the simulation results reported below we set $T_{age} = 70(T_V + T_E)$ and $n_e = 4$ unless stated otherwise.

In the case of the taxi graph, we have a slight drop in the delivery rate (85%-95% depending on n_e). This is because the taxi graph consists of a number of vertices with a small degree (1 and 2), which results in a small mean number of nodes in a stationary regime (less than 4 and 8, respectively) and thus frequent relabeling errors occur in these vertices. This suggests that we could obtain better results if the CPs with a small mean number of nodes were actually considered edges in the CP graph. This is a question for future investigation.

Second, we show in Figure 8, the performance metrics for the grid topology as a function of the square root of the grid size k, for $c = 15$. Figure 8(a) shows the delivery rate. We notice that the delivery rate of the flooding-based approaches (ER, PROPHET) is essentially 100%. The delivery rate of IH drops slightly, but remains very close to 1. This slight drop-off is due to relabeling errors in COGRAD, as described earlier.

Figure 8(b) shows the end-to-end delivery delay. Although the delay of IH is higher than that of flooding-based approaches (around 1.5 times higher), the figure suggests that it is only a small constant factor higher than ER/PROPHET as a function of network size.

Figure 8(c) shows clearly the main advantage of our scheme: it requires a significantly lower overhead per message than ER/PROPHET.

Third, we show simulations based on the inferred city graph in Section III. In these simulations, we vary the parameter c to explore the sensitivity of IH+COGRAD to small values of c. We see in Figure 9(b) that small values of c are problematic; this is because vertices empty too frequently, leading to a high packet drop rate due to relabeling errors.

Finally, Figures 9(b) and 9(c) confirm our finding that IH+COGRAD achieves a very favorable tradeoff, with a delay close to that of flooding-based approaches that are essentially the lowest possible delay, with much lower transmission overhead, which implies that a network operating under IH+COGRAD has a capacity gain of more than an order of magnitude over ER/PROPHET for the scenarios considered here.

This favorable tradeoff is possible because our scheme tightly controls the copies of a message en route, immediately killing any message that strays from the shortest path towards the destination. In flooding-based approaches, messages diffuse throughout the network; in particular, it is difficult in these approaches to ensure that all copies of a message get discarded after one copy of the message has been delivered to the destination. This problem does not arise in IH.

VII. Discussion and Conclusion

We have argued in this paper that node distributions that are heterogeneous in space and relatively stable over time provide an opportunity for mobility-assisted forwarding in partitioned networks, because the underlying topology of concentration points (CPs) and flows of nodes between the resulting islands of connectivity can be learned and used to make progress towards the destination without flooding the network with too many copies of a message.

We have shown that in the presence of stable CPs, our approach significantly outperforms approaches where copies of messages are made without the benefit of an underlying topology. Our approach achieves end-to-end delays of the same order as aggressive flooding-based approaches, but with up to an order of magnitude fewer transmissions per message. This benefit comes from the ability to decide, at every vertex, whether a particular copy of a message has made progress towards the destination or not; we can realize this benefit *even if the mobility process is not predictable* (random walk), and *even if the CP graph is not known a-priori* (thanks to COGRAD).

To do this, we had to take a significant detour, and first develop a scheme to discover this topology. We have shown that under the assumptions of our mobility model, it is - somewhat surprisingly - possible to achieve this by processing the changing set of neighbors of each node, without relying on any explicit signal from the environment or from dedicated infrastructure.

Once we inferred have the CP graph, we can try to forward messages along the shortest path of islands towards the destination. Although we cannot, of course, control the movement of individual nodes, we can nevertheless make progress towards the destination by making a few copies, and letting only those copies survive that go in the right direction.

For this to work, a message has to be able to locate the destination in the CP graph; for this, we have adopted the idea of last encounter routing described in [19], but here locations are vertex labels rather than geographic coordinates. This allows a message to have an estimate of its destination's

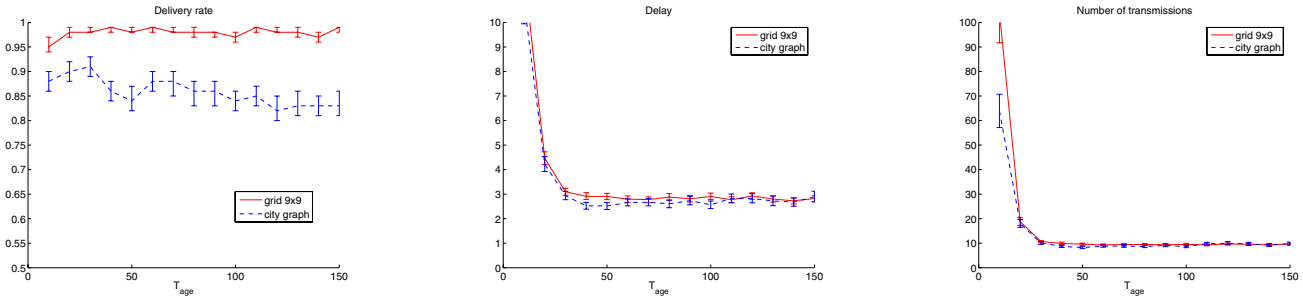


Fig. 6. The grid 9×9 and the city CP graph; as a function of the COOGRAD parameter T_{age}; $n_e = 4$, $c = 15$. (a) Delivery rate, (b) delay, (c) number of transmissions.

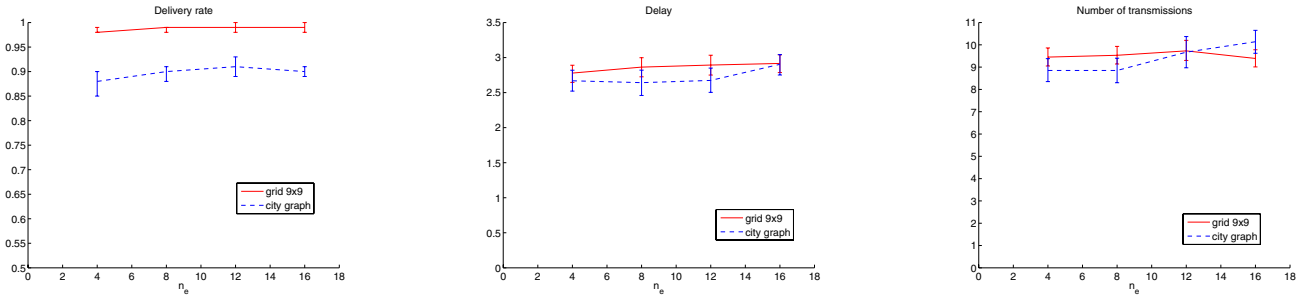


Fig. 7. The grid 9×9 and the city CP graph; as a function of the COGRAD parameter n_e; $T_{age} = 70(T_V + T_E)$, $c = 15$. (a) Delivery rate, (b) delay, (c) number of transmissions.

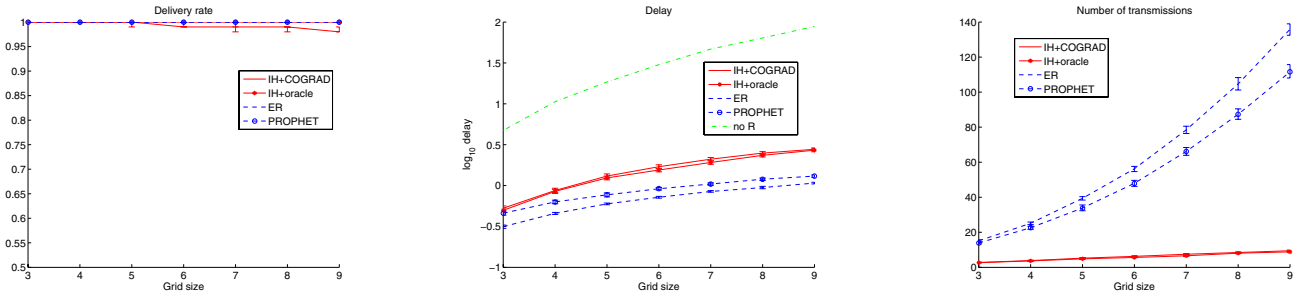


Fig. 8. The grid topology; as a function of k (grid size $k \times k$); $c = 15$. (a) Delivery rate, (b) delay, (c) number of transmissions.

current location; the precision of this estimate tends to improve as the message approaches the destination.

Our simulation results show that our approach significantly outperforms schemes that do not explicitly exploit topological information. Of course, this advantage depends on the presence of a stable topology of concentration points; otherwise, it is probably hard to achieve significantly better performance than schemes such as ER and PROPHET. However, we believe that heterogeneous and stable node distributions tend to be the norm rather than the exception, and we hope to establish this claim through further study of a diverse and representative set of mobility traces. The key point we make in this paper is that stable heterogeneity is beneficial, as it provides structural clues that can be exploited by routing and mobility-assisted forwarding algorithms.

Many questions remain open, and the possible extensions of this work are plentiful. For example, the model of the CP graph, where each vertex is a CP a-priori, and where nodes en-route between CPs cannot communicate with any other node, is certainly somewhat of an exaggeration. We believe, however, that IH and COGRAD can be made more robust so that they will operate even if these assumptions are relaxed. For example, there may be a spectrum of CPs with very different node densities; we envision that a CP that is populated too sparsely, i.e., empties too frequently, should not "qualify" as a vertex in the CP graph, as frequent relabelings are problematic.

Another assumptions concerns our mobility model. It can be expected that COGRAD performance would suffer in the presence of much group mobility, i.e., a set of nodes traveling together for a period of time. Such a group of nodes could "overpower" other nodes when they arrive at a vertex, and impose their label, although they have moved. We think that this is an interesting direction for further research. In particular, it should be possible to improve COGRAD by using history; for example, it may be possible to *repair* labels upon such errors instead of simply relabeling, using history in nodes.

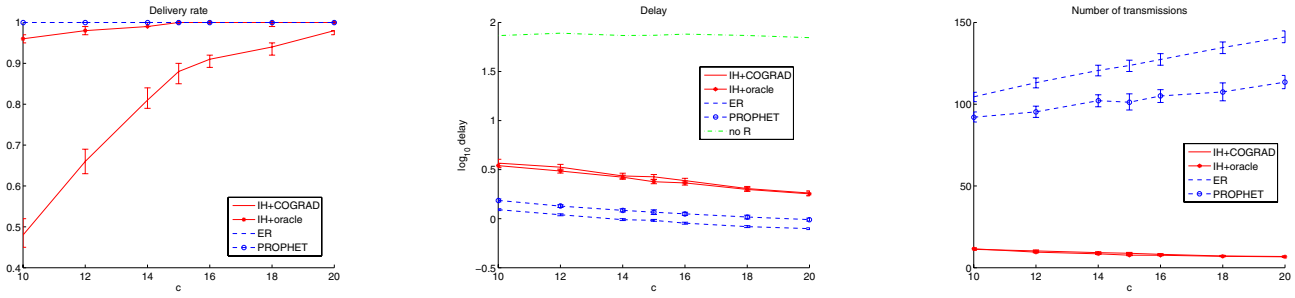


Fig. 9. The city CP graph; as a function of c. (a) Delivery rate, (b) delay, (c) number of transmissions.

ACKNOWLEDGMENTS

The authors would like to thank Henri Dubois-Ferrière, Daniel R. Figueiredo, Maciej Kurant, Hung Nguyen, and Dominique Tschopp for valuable feedback and discussions on this paper. We also thank Holly Cogliati for help in improving the manuscript. We are indebted to the MPT Radio Taxi company in Warsaw, Poland, for making the GPS database available to us.

The work presented in this paper has been supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5006-67322.

REFERENCES

[1] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.S.Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," in *Proc. ASPLOS-X 02*, San Jose, CA, October 2002.

[2] "UMassDieselNet: A Bus-based Disruption Tolerant Network," http://prisms.cs.umass.edu/diesel/.

[3] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket Switched Networks and Human Mobility in Conference Environments," in *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. New York, NY, USA: ACM Press, 2005, pp. 244–251.

[4] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proc. SIGCOMM '03*, August 2003.

[5] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *Proc. ACM Mobihoc '04*, Tokyo Japan, May 2004.

[6] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant networking," in *Proc. SIGCOMM '04*, August/September 2004.

[7] T. Camp, J. Boleng, , and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002.

[8] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing (Tomasz Imielinski and Hank Korth, eds.)*, pp. 153–181, 1996.

[9] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks technical report cs-200006," Duke University, Tech. Rep., April 2000.

[10] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic routing in intermittently connected networks," *Mobile Computing and Communications Review*, July 2003.

[11] X. Chen and A. L. Murphy, "Enabling disconnected transitive communication in mobile adhoc networks," in *Proc. Workshop on Principles of Mobile Computing '01*, August 2004.

[12] M. Musolesi, S. Hailes, and C. Mascolo, "Adaptive routing for intermittently connected mobile ad hoc networks," in *Proc. IEEE 6th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM'05)*, June 2005.

[13] H. Wu, R. Fujimoto, R. Guensler, and M. Hunter, "Mddv: A mobility-centric data dissemination algorithm for vehicular networks," in *Proc. ACM Workshop on Vehicular Ad Hoc Networks (VANET)*, October 2004.

[14] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and Wait: an efficient routing scheme for intermittently connected mobile networks," in *Proc. Workshop on delay tolerant networking and related networks (WDTN-05)*, August 2005.

[15] J. Kang, B. Steward, W. Welbourne, and G. Borriello, "Extracting a Places from Traces of Locations," in *WMASH'04: Proceeding of the 2nd ACM international workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, 2004.

[16] W. Hsu, K. Merchant, H. Shu, C. Hsu, and A. Helmy, "Preference-based Mobility Model and the Case for Congestion Relief in WLANs using Ad Hoc Networks," in *VTC2004-Fall: Proceeding of the 60th IEEE VTC*, 2004.

[17] D. Tang and M. Baker, "Analysis of a Metropolitan-Area Wireless Network," *Wireless Networks*, vol. 9, pp. 107–120, 2002.

[18] J. Dickinson, Gibbons, and S. Chakrabort, *Nonparametric Statistical Inference*. Marcel Dekker, March 2003.

[19] M. Grossglauser and M. Vetterli, "Locating Mobile Nodes with EASE: Learning Efficient Routes from Encounter Histories Alone," *IEEE/ACM Trans. on Networking*, vol. 14, no. 3, June 2006.